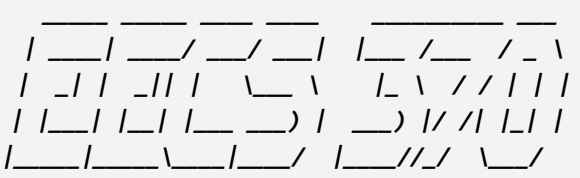
Final Exam



EECS 370 Fall 2022: Introduction to Computer Organization

You are to abide by the University of Michigan College of Engineering Honor Code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam,
nor have I concealed any violations of the Honor Code.

Signature: _____

Name: _____

Uniqname: _____

First/Last name of person sitting to your ***Right***
(Write ⊥ if you are at the end of the row) _____

First/Last name of person sitting to your ***Left***
(Write ⊥ if you are at the end of the row) _____

Exam Directions:

- You have **120 minutes** to complete the exam. There are **10** questions in the exam on **18** pages (double-sided). **Please flip through your exam to ensure you have all 18 pages.**
- You must show your work to be eligible for partial credit.
- Write legibly and dark enough for the scanners to read your answers.
- **Write your uniqname on the line provided at the top of each page. Do this at the beginning of the exam; you will NOT be given time to do it at the end.**

Exam Materials:

- You are allotted **one 8.5 x 11 double-sided** note sheet to bring into the exam room.
- You are allowed to use calculators that do not have an internet connection. All electronic devices with an internet connection are strictly forbidden.

1. True or False	**_____ /**	**12 pts**
2. Branch Predictor	**_____ /**	**5 pts**
3. Pipeline I: Stalls and Forwarding	**_____ /**	**8 pts**
4. Pipeline II: Performance	**_____ /**	**10 pts**
5. Caches I: 3C	**_____ /**	**12 pts**
6. Caches II: Synthesize	**_____ /**	**10 pts**
7. Virtual Memory I: Simulation	**_____ /**	**10 pts**
8. Virtual Memory II: Hierarchy	**_____ /**	**12 pts**
9. Caches III: Reverse Engineer	**_____ /**	**13 pts**
10. Pipeline III: Datapath Modification	**_____ /**	**8 pts**
TOTAL	**_____ /**	**100 pts**

1.	**True or False**	**[12 pts]**
	Complete the following true or false questions	

(a) A processor with lower CPI will always execute a program faster than a processor with higher CPI.
○ True
● False

(b) The number of LRU bits required for a set associative cache depends on cache block size.
○ True
● False

(c) The BTB (Branch Target Buffer) must have an entry for every branch instruction.
○ True
○ False

(d) A multi level page table can take up more memory space than a single level page table.
● True
○ False

(e) Virtual memory allows processes to have a larger address space than physical memory.
● True
○ False

(f) Physically addressed cache reduces TLB accesses compared to virtually addressed cache.
○ True
● False

(g) Reducing a processor's frequency increases main memory latency measured in terms of processor cycles.
○ True
● False

(h) Operating system disallows two processes from ever sharing the same physical page at any given time.
○ True
● False

(i) For a given cache geometry (cache size, block size, associativity), a physically indexed cache is likely to require smaller tags than a virtually indexed cache.
● True
● False

(k) Pipelining improves performance by reducing latency of an instruction.
○ True
● False

(l) For a given program, memory accesses to its page table tend to exhibit lower spatial locality than memory accesses to its data.
● True
● False

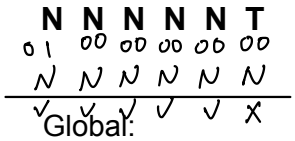
(m) Tags in TLB are derived from physical page numbers.
○ True
● False

2.	**Branch Predictor**	**[5 pts]**

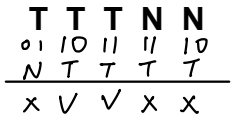
You are given the branch histories of three branches (A, B, and C) as well as the global branch history. A global branch predictor does not keep a separate history record for each branch. Instead, it keeps a shared history of all branches.

(T: Taken, NT: Not Taken)

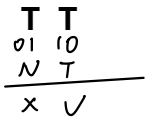
Branch A:



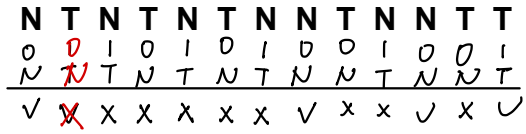
Branch B:



Branch C:



Global:



Determine the number of branches predicted **correctly** for each of the following predictors.

[3 pts] Local (i.e. per branch instruction) 2-bit predictor initialized to "weakly not taken"

i) branch A # of correct predictions 5

ii) branch B # of correct predictions 2

iii) branch C # of correct predictions 1

[2 pts] Global 1-bit predictor initialized to "not taken"

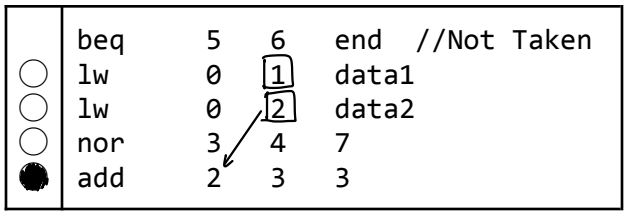
iv) All branches (global) # of correct predictions ~~5~~ 4

3.	**Pipeline stalls and forwarding**	**[8 pts]**
	Determine data hazards and avoidance methods in a pipeline	

Consider a **5-stage** LC2K pipeline datapath that uses **detect and forward** to resolve data hazards, **detect and stall** to resolve control hazards (no branch prediction), and has **internal forwarding** for its register file.

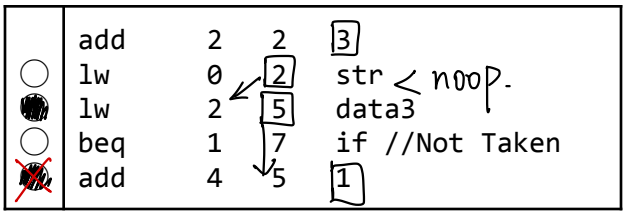
Determine the number of pipeline stalls for each of the following benchmarks. Also, specify the instructions that received forwarded data by shading the circles. **Ignore and do NOT specify instructions that received data through register internal forwarding. file.**

Benchmark 1:



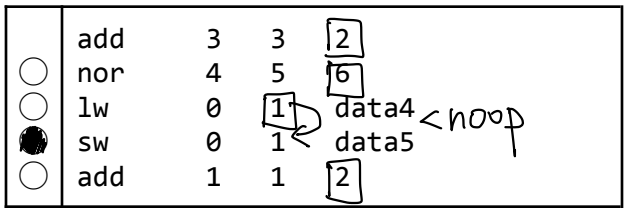
# of Stalls : 3

Benchmark 2:



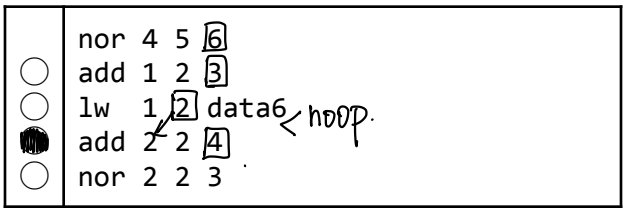
# of Stalls : 4

Benchmark 3:



# of Stalls : 1

Benchmark 4:



# of Stalls : 1.

4.	**Pipeline Performance**	**[10 pts]**
	Perform performance calculations on a given pipeline with a cache	

Consider the following 5-stage LC2K pipeline:

- **Detect-and-forward** is used to handle data hazards.
- **Speculate-and-squash** is used to handle control hazards. Branches are always predicted **"Not Taken"**
- When a lw or sw instruction accesses the memory system in the MEM stage, either
  - There is a **cache hit** and the pipeline **does not stall**
  - or, there is a **cache miss**, which causes the pipeline to **stall for 100 cycles** while the main memory is accessed
- Assume all instruction fetches from an instruction cache are cache hits (no extra stalls)

Assume we run a program with the following characteristics:

lw	25%
sw	15%
add/nor	40%
beq	20%

- 40% of instructions that write to a register (**lw, add, nor**) are immediately followed by an instruction that depends on it
- 10% of instructions that write to a register (**lw, add, nor**) are immediately followed by an independent instruction, and then immediately followed by a dependent instruction.
- 40% of branches are taken
- Cache hit rate is H% (H is a variable)

a) Complete the equation for CPI below using data given above. If you need to use cache hit rate in the equation, refer to it as a **variable H.** **[3 pts]**

CPI = 1

\+ $0.2 \times 0.4 \times 3 = 0.24$ (increase due to control hazards)

\+ $0.25 \times 0.4 \times 1 = 0.1$ (increase due to data hazards)

\+ $(0.25 + 0.15) \times (1 - H) \times 100$ (increase due to cache misses)

0.4

1+ 0.24+0.1 + 40(1-H) = 5.34.

uniqname: _____

1+ 0.34 +40 - 40H = 5.34.

1+40 - 5 = 40H $H = \frac{36}{40} = \frac{9}{10} = 90\%$

b) When we run this program through the above LC2K pipeline, we get a **CPI of 5.34**.

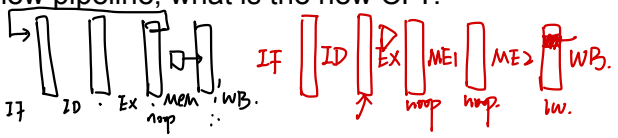
What is the **cache hit rate H**? **[2 pts]**

Cache hit rate H (in percent) = 90%

c) Say we increase the cache size, and that increases the **cache hit rate to 99%**. The latency of this larger cache increases to 2 cycles. To accommodate this change, we split the MEM stage into 2 stages - MEM1 and MEM2 - such that: **[5 pts]**

- lw and sw instructions finish accessing memory at the end of MEM2 stage
- Branches are resolved in the MEM1 stage

If we run the same program from part (a) on our new pipeline, what is the new CPI?



CPI = 1

\+ 0.2x0.4x3 = 0.24 ✓ (increase due to control hazards)

\+ ~~0.25x0.4x2 = 0.2~~ 0.25x0.4x2 + 0.25x0.1x1 = 0.2 + 0.075 = 0.275. (increase due to data hazards)

\+ (0.25+0.15)x0.01x100 = 0.4 ✓ (increase due to cache misses)

0.4

= ~~1.84~~ 1+0.24 + 0.275 +0.4 **(final answer)**

1.24 + 0.675.

1.865.

1+ 0.24 + 0.2 + 0.4

= 1 + 0.44 + 0.4

5.	**Caches: 3C**	**[12 pts]**
	Classify cache misses for a given cache	

Consider a 370 cache in a byte-addressable ISA with the following parameters:

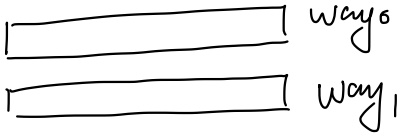
Size: **64 bytes** *(excluding overhead)*
Associativity: **2-way**
Block size: **16 bytes**
Replacement policy **LRU**

64 ÷ 16 = 4 blocks.

16 → 24 set 0

a) 370 cache has 4 cache line(s)

b) 370 cache has 2 set(s) set 1



c) Set index size is 1 bit(s)

d) Block offset size is 4 bit(s)

Assume the cache is initially empty.

Note: Address sequence for parts (e) and (f) are the same.

e) For the following address sequence, simulate infinite and fully-associative caches with appropriate size and block size to help classify the misses (part **f**) for the 370 cache.

The tags are the same for both FA caches. **Fill in ALL the columns.** **[4 pts]**

Address (block offset: 4)	**Tag (in Hex)**	**Infinite Fully-Associative Cache (Hit / Miss)**	**Fully-associative cache (Hit / Miss)** (4)
0x0A	0	Miss	Miss
0x37	3	Miss	Miss
0x53	5	Miss	Miss
0x76	7	Miss	Miss
0x20	2	Miss	Miss
0x0F	0	Hit	Miss
0x3D	3	Hit	Miss

0 3 5 7 2

0 3 5 7
2 0 3

f) Simulate the same address sequence as above for the 370 cache. Classify the misses using the hit/miss information from part (e). **Fill in ALL the columns**. **[6 pts]**

	Address	Tag (In binary)	Set Index (In binary)	Block Offset (In hex)	Hit	Compulsory Miss	Capacity Miss	Conflict Miss	
0000	0x0A	0	0	0xA	○	●	○	○	Miss
0011	0x37	1	1	0x7	○	◐	○	○	Miss
0101	0x53	2	1	0x3	○	◐	○	○	Miss.
0111	0x76	3	1	0x6	○	◐	○	○	Miss.
0010	0x20	1	0	0x0	○	◐	○	○	Miss.
0000	0x0F	0	0	0xF	◐	○	○	○	Hit.
0011	0x3D	1	1	0xD	○	○	◐	○	Miss.

0

1

---

x 3

2 1

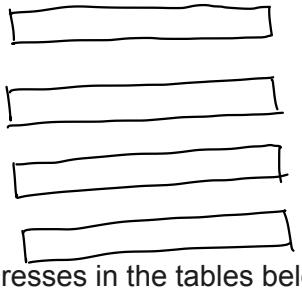
6. Caches: Synthesize address sequences [10 pts]

Run multiple cache accesses and determine when misses occur in different caches

Consider two caches for a byte-addressable ISA:

Cache FA: fully associative, LRU replacement
Cache DIR: direct mapped

Both have 4 cache lines and use a block size of 4 bytes.



Assume they are initially empty. Assume that the selected addresses in the tables below are executed sequentially from left to right.

Note: (a) and (b) below use different address sequences.

a) Determine the **shortest** address sequence (least number of bubbles selected) that will result in a **hit for the second access** to **0x15** in Cache FA, but a **miss** in Cache DIR. Fill out the bubbles for the addresses you select in your sequence. **[4 pts]**

Address (Hex)	0x15	0x38	0x5B	0xA3	0x31	0x24	0x11	0x5B	0x15
Set in Cache DIR	1	2	2	0	0	1	0	2	1
Tag in Cache DIR	0x01	0x03	0x05	0x0A	0x03	0x02	0x01	0x05	0x01
Tag in Cache FA	0x05	0x0E	0x16	0x28	0x0C	0x09	0x04	0x16	0x05
Access Made?	●	○	○	○	○	◍	○	○	●

b) Determine the **shortest** address sequence (least number of bubbles selected) that will result in a **hit for the second access** to **0x33** in Cache DIR, but a **miss** in Cache FA. Fill out the bubbles for the addresses you select in your sequence. **[6 pts]**

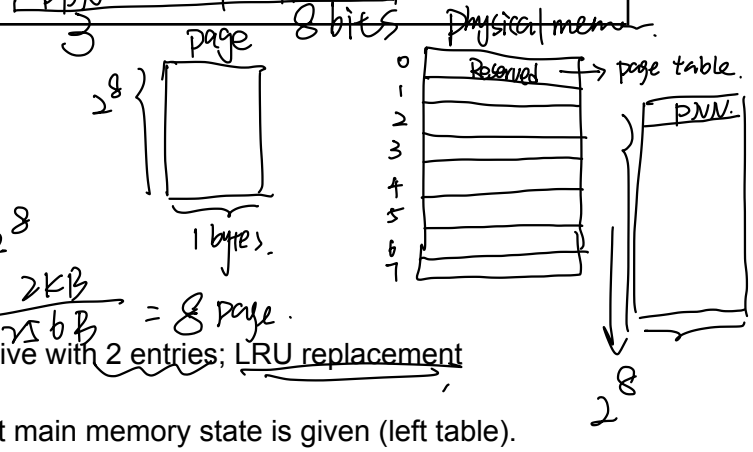
Address (Hex)	0x33	0x38	0x16	0xA3	0x31	0x24	0x11	0x5B	0x33
Set in Cache DIR	0	2	1	0	0	1	0	2	0
Tag in Cache DIR	0x03	0x03	0x01	0x0A	0x03	0x02	0x01	0x05	0x03
Tag in Cache FA	0x0C	0x0E	0x05	0x28	0x0C	0x09	0x04	0x16	0x0C
Access Made?	●	◍	◍	○	○	◍	○	◍	●

virtual address: VPN (8 bits) | offset (8 bits)

uniqname: _____

7. Virtual Memory Simulation [10 pts]

physical address: PPN (3 bits) | offset (8 bits)



Consider the following system:

- 16 bit byte-addressable ISA
- No cache
- Single-level page table
- Page size: 256 Bytes (2^8)
- Physical memory size: 2 KB
- LRU for page allocation
- TLB: Fully-associative with 2 entries; LRU replacement

$\frac{2KB}{256B} = 8$ page.

Assume only one process is executing. The current main memory state is given (left table). Simulate VM and complete the right table for a sequence of virtual addresses issued in order.

Assumptions:

If several unallocated physical pages are available, choose the one with the smallest PPN.
Current state of TLB is empty.
OS has "pinned" page table, so it cannot be evicted from physical memory

Initial Physical Memory State:
(Blank pages are unallocated)

PPN	Contents
0x0	Reserved for OS
0x1	Page table
0x2	VPN: 0xA0
0x3	VPN: 0x7E
0x4	
0x5	
0x6	VPN: 0x21
0x7	

(Left table: NOT graded)

TLB

0x21	0x6
0x7E	0x3
0xA0	0x2

Complete the right table below.

Time	Virtual Address	VPN (in Hex)	TLB Hit? (Y/N)	Page Fault? (Y/N)	PPN (in Hex)
0	0x7E52	0x7E	N	N	0x3
1	0xA050	0xA0	N	Y	0x2
2	0x7E50	0x7E	Y	N	0x3
3	0x21CD	0x21	N	N	0x6
4	0xA0FF	0xA0	N	N	0x2

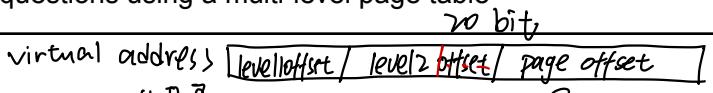
(Right table: Graded)

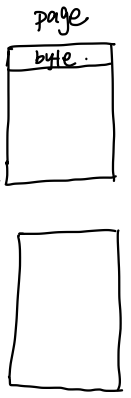
8. Hierarchical Page Table [12 pts]

Answer virtual memory questions using a multi-level page table

Consider the following system:

20 bit ISA		**N-level page table**	N is unknown
Physical memory size:	32KB		
		Page size:	256 bytes
No caches.		Size of any page table***	256 bytes (1 page)
No TLB.		Page table entry size	Unknown





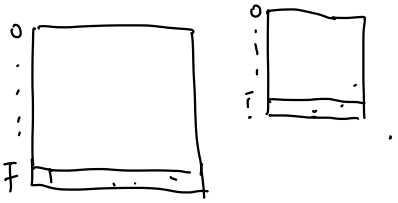
***** Note: 2nd or higher level may have several page tables in each level. Each of those page tables is 256 bytes (1 page) in size.**

(a) Page offset size = _____8_____ bits

(b) Physical page number size = _____7_____ bits

Virtual page number size = _____12_____ bits

(c) OS starts a process P. It creates the 1st level page table and stores it in physical page number (PPN) 0. Initially, all virtual pages for P are unmapped.



Process P accesses the following virtual addresses:

Time	Virtual Address	Page Fault?
0	0xFFF15	Yes
1	0x0000C	Yes
2	0xFFF20	No

If the system made a total of 5 page table lookups (i.e., 5 main memory reads to lookup page tables) while translating the above three addresses, how many lookups were incurred for each?

Hint: If an unmapped page-fault is discovered after looking up an intermediate level page table, no further higher level page-table lookups will be made.

i) 0xFFF15 incurs _____1_____ page table lookup(s)

ii) 0x0000C incurs _____1_____ page table lookup(s)

iii) 0xFFF20 incurs _____3_____ page table lookup(s)

d) Given part (c), the system must have ___3___ level(s) of page tables.

e) Assume that every page table has the same number of entries, and that this number of entries is a power of 2.

What must be the size of the 1st level page table index? ___4___ bits

Brief rationale:

2^8

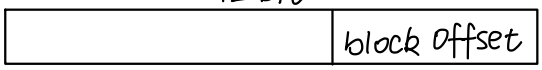
256

f) Given (e), the number of entries in each page table must be 2^4

9.	**Reverse Engineering a Cache**	**[13 pts]**
	Determine a cache's specifications using access patterns	

Assume 12-bit byte addressable ISA.

address



Consider a cache of size 512 bytes. Reverse engineer its other configurations based on the hit/miss outcome for the following address sequence.

Assume LRU replacement. Also, assume that the cache block size and the associativity of the cache are both powers of two. Assume the cache is initially empty.

Access #	Address (Hex)	Address (Binary)	Hit / Miss?
1	0xA65	0b 1010 0110 0101	Miss
2	0x464	0b 0100 0110 0100	Miss
3	0x44B	0b 0100 0100 1011	Miss
4	0x7F6	0b 0111 1111 0110	Miss
5	0xDE1	0b 1101 1110 0001	Miss
6	0x476	0b 0100 0111 0110	**Hit**
7	0x77A	0b 0111 0111 1010	Miss
8	0xA7E	0b 1010 0111 1110	Miss

ⓑ line → 2^9 → 9k bit

assume : full associate. 1 way.

a) Based only on the **first three accesses,** determine the largest possible block size.

Block Size ≤ 2^5 byte(s)

b) Specify which accesses you used to draw the conclusion in part (a)

Access #2 is a miss, and Access #3 is a miss

c) Based only on the **first six accesses**, determine the block size.

Block Size = 2^5 byte(s)

$512 = 2^9$

$2^9 \div 2^5 = 2^4$

d) Specify which accesses you used to draw the conclusion in part (c), also specify whether the access was a hit or a miss.

Access #2 is a hit / miss (circle one), and

Access #6 is a hit / miss (circle one)

e) What is the total number of cache lines in the cache?

 # cache lines = 16 line(s)

f) What is the minimum cache associativity that satisfies the given hit/miss constraints?

(An x-way associative cache is less associative than a y-way associative cache if x < y)

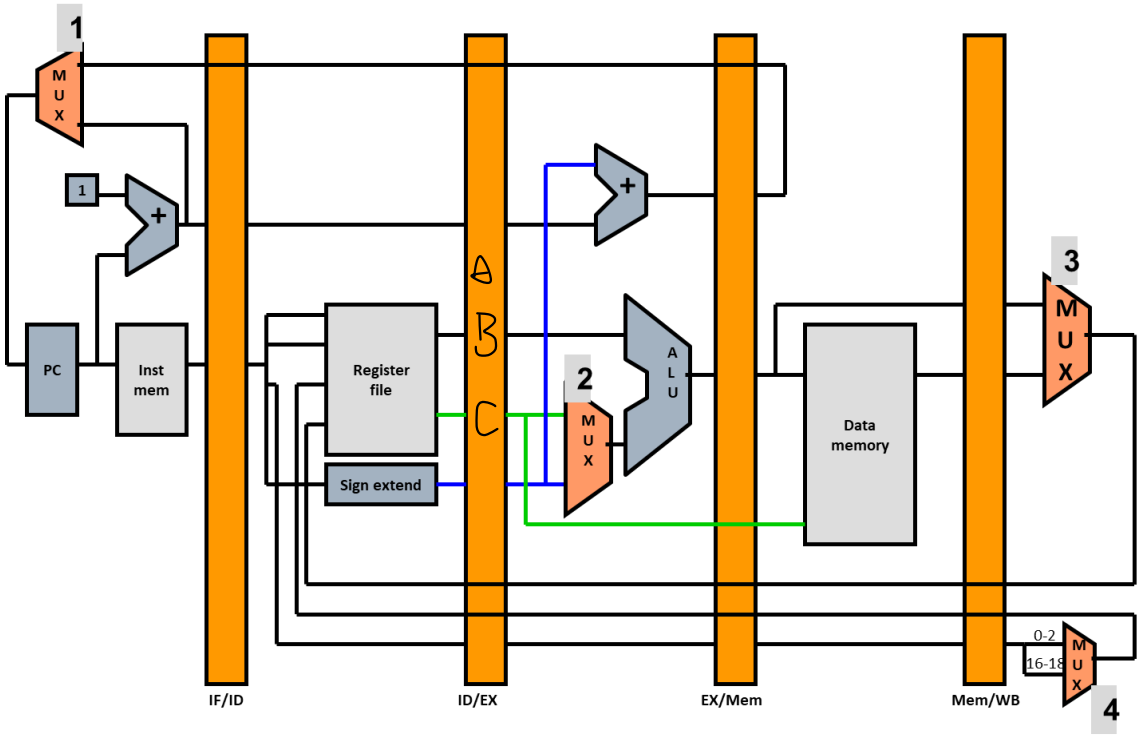
Minimum associativity = 1 **way(s)**

10.	**Pipeline Datapath Modification**	**[8 pts]**

Extend the 5 stage LC2K pipeline discussed in class to support the following **mac** instruction without increasing the number of stages.

$$MAC \quad Reg_A \quad Reg_B \quad Reg_C \quad // Reg_C = Reg_C + (Reg_A * Reg_B)$$

Assume no data-forwarding. Latencies of its components are the same as what we assumed in class.



Use the following additional arithmetic units to support this new instruction.

Component	Delay
1 Multiplier	**1 cycle**
1 Adder	**0.5 cycle**

Describe your new datapath to support the **mac** instruction by answering the following questions.

a) How many register read ports do we need to provision? 3

b) In which stage would you use the new multiplier? (F/D/EX/M/WB) EX

c) In which stage would you use the new adder? (F/D/EX/M/WB) Mem.

d) Which multiplexer (MUX 1 to 4) needs to be extended with an additional input? 3

< Blank Page --- Will **not** be graded>

< Blank Page --- Will **not** be graded>